Adaptability in CORBA: The Mobile Proxy Approach

Benjamin Aziz*
School of Computer Applications
Dublin City University
baziz@compapp.dcu.ie

Christian Jensen
Computer Science Department
University of Dublin, Trinity College
Christian.Jensen@cs.tcd.ie

Abstract

Adaptability is one of the most important challenges in modern distributed systems. It may be defined as the ease with which a software application satisfies the different system constraints and the requirements of users and other applications. Adaptability is needed because distributed systems are inherently open, heterogeneous, and dynamic environments integrating a wide range of platforms, operating systems and applications from a number of different sources.

In this paper, we propose to use mobile proxies to provide adaptability in distributed applications integrated using the CORBA technology. Downloading stubs and skeletons at runtime allows the adaptation of either client or server interfaces as well as the protocol linking the two.

1. Introduction

The rapid advances in computer science have facilitated the development of a wide range of distributed technologies ranging from office-wide LANs to the worldwide Internet. These technologies form a sophisticated environment, in which changes take place rapidly with new applications added, old ones removed or upgraded, and new standards, protocols and data formats emerging all the time. This situation creates the necessity for adaptability among entities interacting in this environment.

At a basic level, *adaptability* may be defined as *the ease with which a software application satisfies the different system constraints and the requirements of users and other software applications* [6]. It describes the adjustment of some entity to the needs and abilities of other entities, thereby changing its state and/or behaviour accordingly.

Adaptability can be a matter of survival where, unless an entity adapts itself to some situation, it cannot continue to exist and interact with other entities. In general, the goal of adaptability is to reach some minimum acceptable level of performance or functionality, known as the Quality of Service (QoS) level [6]. Under any circumstances, adaptability is desirable as it lengthens the lifetime of an entity, offers flexibility, and generally, saves effort, time, and resources.

In this paper, we propose a new mechanism called the *Mobile Proxy* (MP) mechanism, based on the proxy principle [23], as a means of achieving adaptability in distributed applications integrated using the Common Object Request Broker Architecture (CORBA) technology [25].

A proxy is defined as a service representative that resides at the client's site. It provides an interface to the service and takes care of the communication protocol with that service. It also takes care of the marshalling of data, the checking of the validity of calls, and any other low-level processing, thereby making distribution transparent to the client. Due to this transparency, proxies are popular and have been adopted by many distributed systems, including distributed objects technologies [17].

In CORBA, the proxy appears as the *stub*, which is linked statically to the client. The stub sends invocations to the server object via the ORB, which will carry those invocations to another, complementary piece of code linked to the server and known as the *skeleton*.

The MP mechanism explores the possibility of dynamically extending the functionality of the proxy (stub), hence, specialising it to perform additional user-defined functionalities. This situation results in the existence of multiple *adaptable* proxies per service, each carrying a different functionality. The relevant proxy is then chosen according to the requirements of the client-

* This work was undertaken as part of the MSc programme in networks and distributed systems at the Computer Science Department, University of Dublin, Trinity College, 1998-99.

server interaction, where a client will be supplied with the proxy that best suit these requirements.

As an example, the functionality can be data compression, compressing all the incoming and outgoing data at a strategy and speed aimed at improving the overall communication performance. As we shall explain later in the paper, compression may benefit a client running on a slow wireless network (e.g. infrared network) but could degrade the communication performance for a client running on a fast backbone network. Adaptability decision would then choose between using or avoiding the compression functionality.

The MP mechanism relies on code mobility to support dynamic adaptability through proxies downloaded at runtime. Proxy mobility will keep clients as lightweight as possible, since the many proxies will reside at the service site(s). It also promotes the notion of proxy *freshness*, where the technology used to implement the adaptable proxies is guaranteed to be up to date. Moreover, these proxies can be changed by the service administrators without the need to inform or change the clients that use that service. Such a costly change would be inescapable in the case of statically linked proxies that reside at the client sites.

We also introduce the notions of the *Environmental Object*, which describes the different environments and components, and the *Environmental Repository*, which acts as a holder for these objects. We believe that these notions are necessary for the adaptability process to be efficient.

The *Adaptable Proxies* (APrx) system is a prototype that implements the MP mechanism. The system is built using Java [7] and offers a set of APIs for any Java application integrated using CORBA. The evaluation of the results obtained indicates the significance such a system may have on the goal of adaptability.

The rest of this paper is structured as follows: Section 2 provides a review of possible application scenarios where adaptability may be required. The MP mechanism is discussed in Section 3. The APrx prototype is described in Section 4 and an evaluation of this prototype is presented in Section 5. Section 6 describes related work, and finally, Section 7 provides our general conclusions and discusses the prospects of future work.

2. Adaptability scenarios

In this section, we review some examples of scenarios that reflect the need for adaptability and the employment of proxies in achieving such adaptability.

2.1. Communication performance

Communication performance is an important area where adaptability may be applied. The delay incurred in exchanging messages and data between a client and server can be expressed by the following equation:

$$D = T_p + T_x \quad (1)$$

Where D is the communication delay (second), T_p is the processing time (second), and T_x is the transmission time (second).

As (1) indicates, communication delay consists of two times: processing time (T_p) and transmission time (T_x). Processing time refers to any pre-transmission and post-reception processing that may be applied to the data exchanged. Transmission time, on other hand, is simply the time taken by that data while travelling on the network.

The transmission time itself depends on two other factors: the size of transmitted data and the network speed:

$$T_x = S/V \quad (2)$$

Where S is the data size (byte) and V is the network speed (byte/second).

The size is an obvious factor affecting the transmission time linearly. However, network speed is less obvious depending on a number of other factors including, the physical bandwidth of the network(s) traversed by the data, the amount of traffic flowing, and the actual distance travelled.

Consequently, the network speed factor is less controllable (from an application's point of view) than the data size factor. So, from (1) and (2) one may conclude that in order to improve the overall communication delay D, the size of data S has to be reduced:

$$D = T_p + S/V \quad (3)$$

Compression techniques (a pre-transmission process) may be used to minimise the size of data prior to its transmission. After its reception, the data would then be decompressed (a post-reception process) to retrieve the original information.

Ideally, if the compression (decompression) process did not consume any time, i.e. T_p in (3) was 0; the minimum communication delay would be reached at the smallest size:

$$D_{(min)} = T_{x(min)} = S_{(min)}/V \quad (4)$$

However, (4) is not realistic since the compression (decompression) process consumes time, which represents the processing time T_p. At this point, a trade off begins to form as to whether apply compression, hence minimise T_x, or just send the data directly without compressing them and so, minimise T_p. The decision will be driven towards optimising D and, as the results of Section 5 have shown, such a decision will largely depend, on the nature of data.

To better illustrate the scenario, assume a database server that is queried by two clients. The first (Client1) is running on a slow wireless network, whereas the second (Client2) is running on a fast backbone network. The server should supply the former with a *compression-*

proxy and the latter with a *null-proxy* (a proxy that embodies no extra functionality). Figure 1 illustrates the idea, where Π denotes compression and Ξ denotes decompression (or vice versa, depending on the direction of the transmitted data).

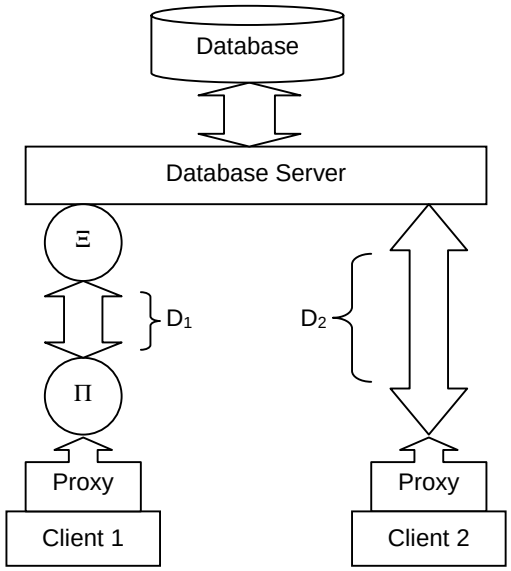

Figure 1. Communication performance adaptability

Assuming there are no other processing times apart from the compression/decompression time, and from (3), we will reach (5) for the first and second clients respectively as:

$$D_1 = T_{compression} + T_x$$
$$D_2 = T_x \quad (5)$$

In case no compression was employed, D_1 would become D_2, which is only composed of the transmission time T_x. Now, the size of data at which D_1 starts getting less than D_2 is known as the *breakeven point*. Intuitively, breakeven point is the point at which compression starts improving D_1 ($D_1<D_2$). Ideally, this point would be at 0 bytes. However, in reality it could be more than that, depending, among other things, on the type of network the client is running on. For example, in high-speed networks, it can reach several hundred KB or even a few MB.

2.2. Communication security

Adaptability is also desirable in the context of security. Assume a server that is interacting with a client running on (or separated by) a foreign network that cannot be trusted. This will lead to the exposure of any communicated data to all sorts of attacks aimed at compromising the privacy, integrity, and authenticity of that data. Therefore, appropriate measures are required like, encryption, digital signatures (certificates), and message digests to protect the data from such attacks. The decision as to what functionality to employ depends, in the first place, on the criticality of the transmitted data and the required security properties.

For example, to preserve privacy, data will have to be encrypted prior to its transmission using, either a public-key protocol like RSA [20], or a secret-key protocol like DES [14]. On the other hand, if privacy is not as important an issue as the authenticity of data, digital signatures using DSA [15] may be preferred to full encryption.

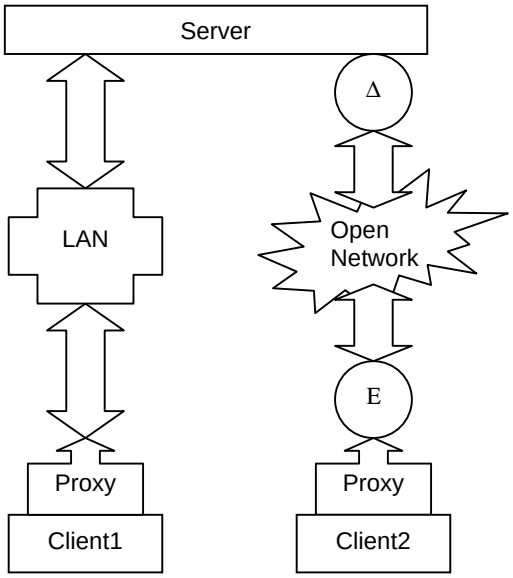

Figure 2. Communication security adaptability

Figure 2 illustrates how a server may supply an *encryption-proxy* to Client2 that is separated from the server by an open network. On the other hand, no security is required for data travelling within a LAN, since the network is closed and consequently, assumed trusted. Therefore, a null-proxy would be appropriate for Client1. The 'E' denotes encryption, whereas 'Δ' denotes decryption (or vice versa, depending on the direction of the transmitted data).

2.3. Other scenarios

In addition to communication performance and security, adaptability can benefit other situations where the requirements of the client/server interaction may include the likes of group communication, caching, fault tolerance, load balancing, and others. However, due to shortage in space, we will only discuss the group communication and caching scenarios.

2.3.1. Group communication. So far, we have considered the server as being composed of a single process. This, however, may not always be the case. In some applications, especially where fault-tolerance is sought, the server could be a process group composed of a number of member processes, one of which may be the master process. To communicate with such a group, the master process, normally, assumes the responsibility for forwarding any requests and replies made on behalf of the group.

Multicasting, however, is another way of communicating with the group. A *multicast-proxy* could be employed to, transparently, forward any messages sent by the client to the intended recipients in the group, as in Figure 3. Similarly, the proxy will manage any (possibly duplicate) responses sent back from those members.

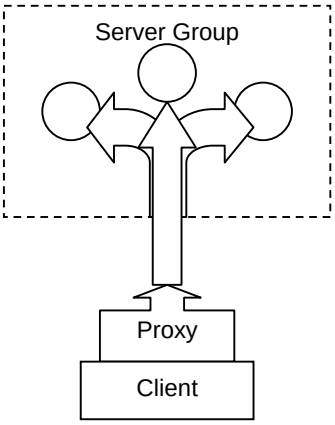

Figure 3. Group communication adaptability

2.3.2. Caching. We have already reviewed one case by which performance can be improved, i.e., optimising the communication delay.

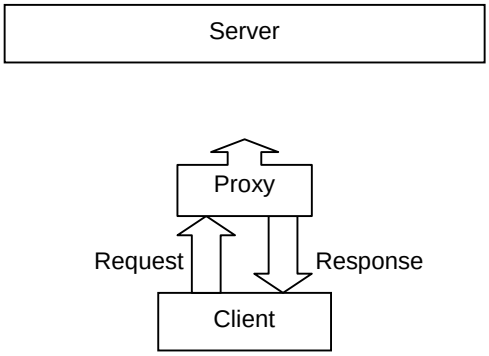

Figure 4. Caching adaptability

Caching is another performance-enhancing method, which is popular in Web proxies [12]. The *caching-proxy* manipulates its local position at the client site and attempts to satisfy any requests, if the requested data are available locally. It is only when such data are not available locally that the proxy will forward the request to the server. This is shown in Figure 4. Caching is useful only in situations where the freshness of data is not crucial to the client.

3. The MP mechanism

Consider a client-server application integrated using the CORBA technology. The server object is defined by a set of one or more interfaces written in a special Interface Definition Language (IDL) [18], which is defined by the Object Management Group (OMG) as part of the CORBA specification. These interfaces reflect the nature of the service offered by that object and define a standard method of interaction.

Before a client can avail of the service offered by a particular server object, it first has to *locate* that object using any of the services defined by CORBA (e.g. Naming or Trader service). Alternatively, the client can create an instance of a proxy object (the stub) either by *binding* to the server or by using Object Reference Strings.

The client may then start invoking methods identified in the server's interface. This is normally done in a static manner, where the invocation is handled by the client stub and delivered to the server skeleton via the Object Request Broker (ORB). Alternatively, if the client does not have knowledge of the server's interface at compile, it can query a special database known as the Interface Repository and obtain information about the server's interface. It can then use the Dynamic Invocation Interface (DII) component of the ORB to make invocations on the server object.

CORBA also allows dynamic server programming through the Dynamic Skeleton Interface (DSI) where invocations are allowed on IDL interfaces for which no server objects exist. These objects are implemented at runtime by the server host.

It should be noted that the dynamic aspects of CORBA applications, i.e. DII and DSI might allow for some form of adaptability to be realised. However, the nature of this adaptability is more of an *interface-adaptability* than *proxy-adaptability*, which is what we propose and will explain in due course.

In the Mobile Proxy (MP) mechanism, we introduce three other entities to a CORBA application that we believe are essential to the goal of adaptability. These are the *mobile proxy*, the *Environmental Object*, and the *Environmental Repository*.

3.1. The mobile proxy

Mobile proxies are essentially CORBA stubs. However, they differ from the normal stubs in two

aspects. First, they are mobile in the sense that clients do not have the mobile proxy code that will be supplied at runtime by the server. Second, they are specialised to perform certain functionalities aimed at achieving adaptability.

Mobility is necessary to maintain lightweight clients and promote the idea of proxy *freshness.* This implies that any changes applied to the internal implementation (and not the advertised interface of the server) of these proxies are not transmitted to clients until runtime. Therefore, obviating the need for changing the clients every time proxies are updated, something that has to be done with statically linked proxies. However, mobility has the potential drawback of compromising the security of the system running the foreign code, and so, access rights must be restricted by a security policy regarding the execution of mobile code.

The specialisation of mobile proxies will be in the context of the adaptability scenarios we already discussed in last section. Each scenario requires the implementation of extra functionalities to be embedded in or linked to a specific proxy. These functionalities will then modify the communication session according to the requirements of that particular scenario. Such requirements are described by the second entity: the Environmental Object.

3.2. The Environmental Object

An Environmental Object is a data structure that holds a description of the different environments in which clients and servers are running, the requirements of these clients and servers, and any other runtime demands. They may be described as a form of Meta Objects [13] that carry information about (and at a higher level than) their application.

An Environmental Object will hold necessary information about its owner. Such information may include the available bandwidth to the owner, the current and home addresses of the owner, whether the owner is a single or a group process, and whether the owner is sensitive or not to the freshness of any data requested. The amount of information an Environmental Object will hold depends on the complexity of its owner and how important the adaptability issue is to that owner. The more information contained the better the description of the environment and its needs, and consequently, a more precise adaptability decision will be reached. This decision is crucial because it will eventually determine the success or failure of the whole adaptability goal.

3.3. The Environmental Repository

An Environmental Repository represents a special database designed to hold the Environmental Objects of the different server environments. The repository will also hold bindings between the Environmental Objects and the names of the servers to which they belong. Additionally, it may be authorised by a server to hold the library of mobile proxies that represent that server. All this can be done through a registration process similar to the one carried out for an Implementation Repository.

In addition to being a holder, an Environmental Repository will also perform a decision-making process, prior to the client/server interaction session, during which it will compare the different client-server Environmental Objects and produce a decision as to which mobile proxy should be used in a particular scenario. This decision may be the mobile proxy itself, in which case it will be sent directly to the client.

A network may contain any number of Environmental Repositories depending on the number of server objects maintained by the ORB and the size of the network

3.4. The runtime dynamics

We will proceed to describe the working details of the MP mechanism:

First, a client willing to adapt to the server will commence by contacting the Environmental Repository and sending it the name of the required server and the Environmental Object of the client, as shown in step 1 of Figure 5.

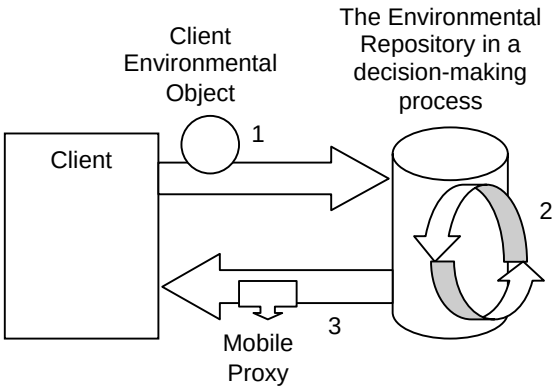

Figure 5. The client contacts the Environmental Repository

The Environmental Repository then undergoes a decision-making process (step 2) during which it may use any techniques ranging from the simple if-then rules, to Case-Based Reasoning (CBR). This process will compare the different requirements of the client and the server and of their environments (as represented by the Environmental Objects) and reach a decision as to the most suitable functionality (if any). The repository will then send a mobile proxy (step 3) embodying that functionality to the client or, alternatively, send the address from which that proxy may be downloaded, if it

is not available at the repository. The repository should also be able to inform the server of the adaptability decision so that the latter can employ the appropriate functionality that complements the proxy's functionality (e.g. decompression versus compression).

After downloading the mobile proxy, the client can now start making invocations on the server object. All the data sent or received in the communication session may now be modified according to the functionality of the proxy. This is illustrated in Figure 6, where "MP" stands for the mobile proxy and "F" for the complementary functionality at the server.

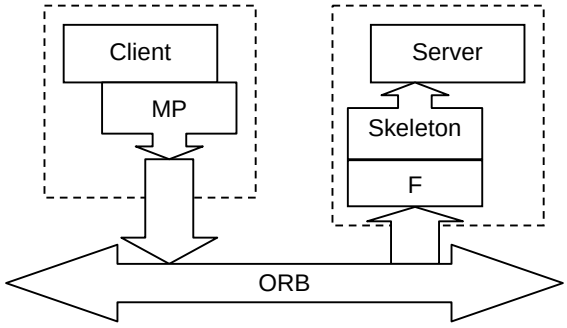

Figure 6. The client invoking the server object

4. The Adaptable Proxies system (APrx)

The APrx system is an experimental implementation of the MP mechanism built for Java applications that are integrated with CORBA. In version 1.0 of the system, JDK 1.1 was chosen as the language of implementation and OrbixWeb v3.1c [16] as a sample ORB. The system was implemented as part of the Mobile Proxies project [2].

In general, the APrx system is divided into five logical modules as shown in Figure 7 below. These modules are (with respect to the numbers shown in the figure):

1. The client adapter
2. The server adapter
3. The ORB adapter
4. The proxy loader
5. The class loader

The five modules constitute the core of the APrx system and together, encompass ten Java classes and one interface.

The c*lient adapter* module is composed of two Java classes that represent a front-end API to the client object and that will initiate and end the adaptation process. It provides methods for contacting the Environmental Repository and supplying it with an Environmental Object (as described in last section). The client adapter can also inform the server of the adaptability decision through a special method that sends to the server adapter, an object containing the URL of the mobile proxy downloaded by the client. This may be required in case the Environmental Repository is unable to inform the server of the adaptability decision. It also has the advantage that *knowledge* will be distributed between the server and its client, which constitutes a form of security.

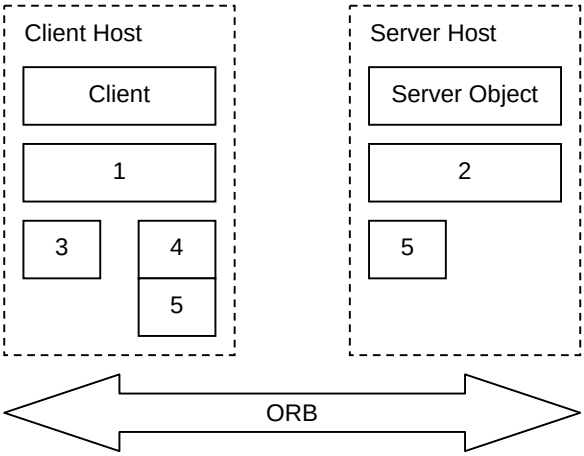

Figure 7. The APrx system modules

For example, if the URL was changed maliciously on its way to the client from the Environmental Repository by a third entity, the server will immediately discover that this URL is not part of its site or the Environmental Repository's site. It can then either inform the client or terminate the communication session, where in both cases, the client will not proceed with executing the malicious code.

The s*erver adapter*, composed of three Java classes, provides an API to the server object, which insures that the functionality adopted by the client side is complemented at the server. For example, if the client opted for compressing all the transmitted data bytes, the server should use decompression to reverse the effect of the client functionality. The module is also responsible for managing concurrency issues that can rise from multiple clients attempting adaptability with the server at the same time. The server adapter will keep the state of every connection opened by a client and the current functionality adopted by that client.

The *ORB adapter* module is composed of two Java classes that are part of the client object, although it is hidden from the latter. It provides methods that will allow the ORB to inform the proxy loader of the point in time when the client needs a proxy because a reference to the server object has entered the client's address space. This is possibly due to the client binding to the server.

The p*roxy loader* module will then initiate the downloading of the mobile proxy code and any additional code employed by that proxy. The module is composed of a single Java class.

Finally, The *Class Loader* module will download the data bytes and maintain some form of security by verifying the digital signatures associated with those bytes. The module is composed of three Java classes and is used by both the client and server objects.

The APrx system also defines a special interface that has to be implemented by every mobile proxy in order for that proxy to be identified as adaptable.

In version 1.0 of the system, the implementations of the Environmental Object and Environmental Repository were left out. We believe that these two entities demand further research due to their complexity and significance.

4.2. The functionality tree

In addition to the core modules, the APrx system defines a *functionality tree*, which imposes a structure on the adaptable mobile proxies. This structure, illustrated in Figure 8, defines the different levels by which these proxies (and any other related code) may be classified. The tree is divided into four levels: The *default proxy* level, the *mobile proxy* level, the *interceptor* level, and the *processing* level.

The default proxy level represents the default stub, which is usually provided by any IDL compiler. The stub does not contain any functionality apart from the normal packing of data.

The mobile proxy level contains the adaptable mobile proxies, which will be downloaded at runtime and will override the default proxy. Any number of these proxies may exist depending on the adaptability scenarios the server is willing to support. In the case of OrbixWeb, these will be the *smart proxies* [16].

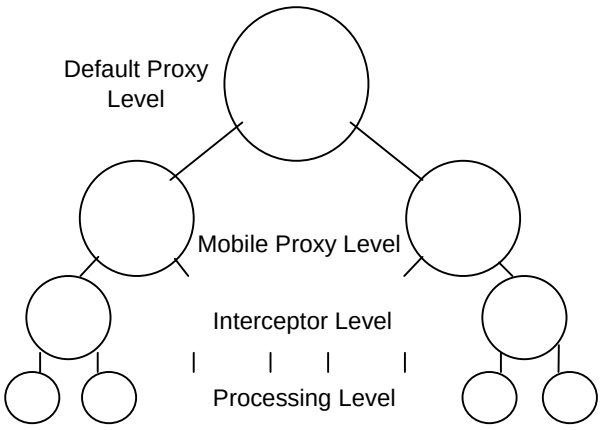

Figure 8. The APrx functionality tree

The interceptor level contains any CORBA interceptors [4] that may be employed by the mobile proxies. These are used primarily to intercept and expose the data entering or leaving the address space of a process and so, allowing for low-level processing like compression and encryption to be applied. These interceptors will most probably be the *transformers* [16] of OrbixWeb, although *filters* [16] are also possible. However, transformers are preferred since they work at a lower level than filters, and, therefore, can expose the communicated data bytes.

The final level, the processing level, defines any other objects that might be needed by the previous two levels to accomplish their expected role. For example, it might include objects that perform Huffman encoding [21], DES, or RSA encryption, and MD5 [19] message digesting. The level is intended to be open in the sense that it can expand vertically to include any number of objects depending on the level of implementation complexity.

4.3. Example

Let's take the scenario of section 2.1 as an example of how the APrx system may improve the communication performance.

Consider again a client running on a slow wireless network. We will assume that the client already has the default stub code, although this is not necessary.

The first step will be for the client and server to create instances of the client and server adapter objects and invoke special methods that will render them ready to adapt each other. A reference to the server object may then enter the client's address space in which case the ORB will inform the client of this by invoking a special method on the ORB adapter.

The client adapter will then proceed to contact the Environmental Repository and supply it with its Environmental Object. This object will include information about how slow the client's network is. The Environmental Repository should reach a decision that the most suitable proxy is the compression-proxy and should inform the client adapter of the URL of this proxy.

The client adapter then will invoke the necessary methods on the proxy and class loaders to download and install that proxy and any transformers and processing objects employed by it. The client adapter will also contact the server adapter, if necessary, and inform it of this decision. This will allow the server adapter to install the necessary decompression transformer and related processing code. The client can now start invoking methods on the server object. All the communication will go through the mobile proxy and associated transformers and processing objects.

5. Evaluation

The evaluation of the APrx version 1.0 system was carried out for the communication performance and security adaptability scenarios.

A test package was developed to measure the different transmission and processing times of the APrx system in both scenarios. The package represented a traditional file transfer application where the client sends a file to the server. Such an application is a good example where the nature of the transmitted data affects the adaptability decision. The design of the test application allows the client to read a file from the local file system, send it to the server object on another host, and receive an acknowledgement. At the same time, the application will register all the necessary transmission and processing times as well as the size of data at different stages of the application.

Tests were carried out on three types of data files transmitted over a 10 Mbps LAN. These included image files in the JPEG, PDF files, and text files.

In the communication performance test, compression was used as a means of improving the overall communication delay. The first set of files employed in the test was the JPEG files. Results revealed that these would not compress by more than 5%, which according to equations (1) and (3) of section 2.1, meant that T_p increased considerably without much reductions in S or T_x, therefore, leading to an overall increase in D. Adaptability in such a situation was to avoid compression and transmit the JPEG file directly.

PDF files, on other hand, had a better compression ratio ranging from 23% to 65%. The breakeven point was at about 187.5 KB as shown in Figure 9. At that size compression would benefit the overall communication performance.

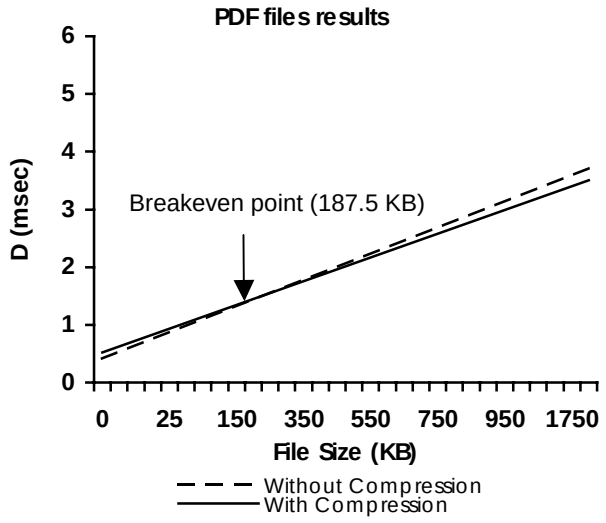

Figure 9. Performance results of the PDF files

Text files had an even better compression ratio reaching over 90% in some cases, and so, a breakeven point of as low as 93 KB was achieved as shown in Figure 10.

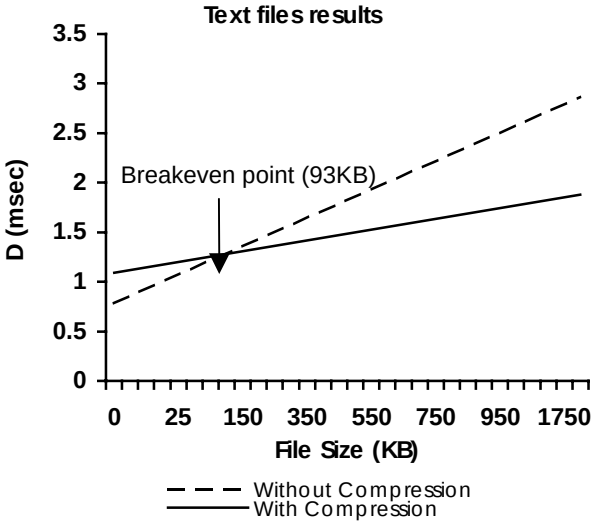

Figure 10. Performance results of the Text files

An analysis was performed to predict the breakeven point for different network speeds. This analysis predicts that slow networks will affect the overall communication performances, where T_x is the dominant component, more than those that have a fair ratio of T_x to T_p, or where T_p is dominant (fast networks). Therefore, it is in the slow networks that compression would be most useful, since transmission time will reduce notably without much increase in the processing time. On the other hand, compression becomes infeasible for high-speed networks.

Table 1 shows how the breakeven point of the PDF files would vary according to the type of network on which the application is running.

Table 1. The breakeven point for different networks (PDF files)

Network	Speed (Mbps)	Breakeven point (KB)
GSM	0.0096	0
ISDN	0.128	0
Ethernet	10	187.5
Telesat Satellite	12	398
T3 and networks with higher speeds	>45	Compression not feasible

The second performance test was dedicated to measuring the expense of three security techniques. These included symmetric and asymmetric encryption and digital signatures. In the case of encryption, the transmitted file was encrypted before transmission and decrypted at its destination. Whereas with digital signatures, the file was signed and the signature attached to the transmitted data. The signature was then verified at the destination.

As shown in Figure 11, asymmetric encryption using RSA was the most expensive, followed by symmetric encryption using DES, and finally, digital signing with DSA. 256-bit keys were used in all three cases.

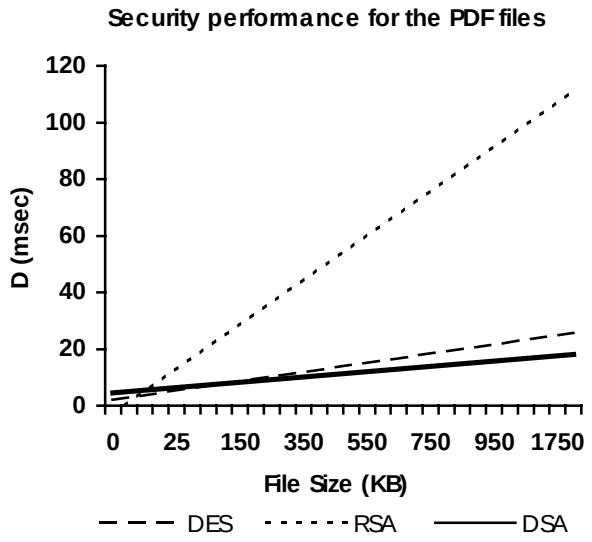

Figure 11. Performance results of security adaptability for PDF files

It is worth noting that these tests were not meant to improve the performance of the application but simply to get an idea of such performance. However, security techniques should be feasible in order for them to be practical and employable.

6. Related Work

Adaptability has been a major concern for software development recently and the following examples are but a few that one can mention, although much work is still to be done in this area if modern distributed systems are to be characterized as adaptable.

The RAPP system [22] is a CORBA-based platform aimed at improving the QoS level by inserting proxy objects in the path of the transmitted data. However, The system is tailored mainly to improve the quality of service, as opposed to the more general adaptability of the APrx system.

The Reliable Multicast proXy (RMX) [3] is another system designed with the goal of adaptability in mind. The RMX system proposes a heavyweight multicast proxy placed on a separate host between a server and its clients. The proxy has a complete knowledge of the state of those clients (e.g. their respective bandwidths), will modify any data transmitted from the server using certain functionalities to adapt to the environments of those clients (e.g. compressing the data for slow clients). The system offers a wide range of functionalities that will cater for different adaptability scenarios.

The possible advantages of adaptability by mobility are evaluated by Ismail and Hagimont [8]. Their proposed system, called the *agent server*, allows adaptability in a pure Java environment. The agent server provides a mechanism for clients to upload mobile agents to the servers. These agents will adapt the server's interface to the operational needs of the client, thus allowing extra functionality to be added to the server. This approach is complementary to our adaptable mobile proxies.

The Jini technology [9] is probably the best known example of adaptability through mobility. Servers register a description of the services they offer with a special lookup server along with a proxy that permits clients to avail of that service. Clients will query the lookup server to learn of available services and obtain the relevant proxies, thereby allowing client/server interaction to be adapted at runtime.

Adaptability in Jini is regarded as a form of interface-adaptability. A client chooses a service (e.g. printer) and a proxy with a suitable interface is given back to that client. This is in contrast to our approach where a service interface is constant but the implementation of the proxies representing that service and carrying the same interface are different.

Other related work includes the LEAD++ language proposed by Amano and Watanabe [1], which is an object oriented language that takes dynamic adaptability into consideration and is based on a reflective model. Also Kidston et al. [11] propose a proxy solution that will improve the QoS level for communication between wired and wireless environments. Adaptability is achieved by allowing a wired environment with high QoS to meet the lower QoS of a wireless environment.

Finally, Katz [10] looks at the issue of adaptation and mobility in wireless information systems where the awareness of the location and situation are taken as forms of adaptability.

7. Future work

An important area where further research should be directed is the area of Environmental Objects and Environmental Repositories. Such research should include the information that will be encoded in the Environmental Objects and the technologies used in building the Environmental Repository.

Another area would the cascading of different mobile proxies in order to realise complex scenarios and enrich the adaptability semantics.

8. Conclusion

The fact that modern distributed systems are composed of a wide range of products meant that these systems should offer mechanisms to allow different applications to interact in a way that will satisfy certain application- and environment-level requirements.

One such mechanism, proposed in this paper, is the MP mechanism. The mechanism aims at providing adaptability in the CORBA framework using mobile proxies.

The MP mechanism also introduces the notions of Environmental Objects, which describe the nature of the application and its environment, and Environmental Repositories, which act as holders of the Environmental Objects. Appropriate support for these notions requires further research.

The versatility of the MP mechanism is demonstrated by the different adaptability scenarios that may benefit from it. The mechanism is implemented by the APrx prototype, which is an open, extensible system that provides a standard set of APIs to Java applications integrated using CORBA. The results of test applications revealed the significance such a system may have on the adaptability goal.

9. References

[1] N. Amano and T. Watanabe, "LEAD++: An Object-Oriented Language Based on a Reflective Model for Dynamic Software Adaptation", in *Proceedings of the 31st International Conference on Technology of Object-Oriented Language and Systems (TOOLS Asia'99)*, December 1999, pp. 41-50.

[2] B. Aziz, "Mobile Proxies", *MSc Dissertation*, Technical Report TCD-CS-1999-60, University of Dublin, Trinity College, Dublin, Ireland, December 1999.

[3] Y. Chawathe, S. McCanne, S. Fink, and E. A. Brewer, "A Proxy Architecture for Reliable Multicast in Heterogeneous Environments", in *Proceedings of ACM Multimedia'98*, Bristol, U.K., September 1998.

[4] CORBA/IIOP v2.3 Specification, the OMG, 1999, Chapter 21.

[5] Downing, T.B., *Java RMI: Remote Method Invocation*, IDG Books Worldwide, Inc., Foster City CA, USA, 1998.

[6] Evans, M.W. and J. Marciniak, *Software Quality Assurance and Management*, John Wiley & Sons, Inc., New York, 1987.

[7] Gosling, J., B. Joy, and G. Steele, *The Java Language Specification*, Sun Microsystems, 1996.

[8] L. Ismail and D. Hagimont, "A performance evaluation of the mobile agent paradigm", in *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'99)*, Denver, CO USA, November 1999, pp. 306-313.

[9] *Jini™ architecture specification*, Sun Microsystems, 1999.

[10] R.H. Katz, "Implementing Communication through "Situation Awareness" Adaptation and Mobility in Wireless Information Systems", IEEE Personal Communications Magazine, Volume 1, Number 1, First Quarter 1994.

[11] D. Kidston, J. P. Black, and T. Kunz, "Transparent Communication Management in Wireless Networks", in *Proceedings of the 7th Workshop on Hot Topics in Operating Systems*, Rio Rico, Arizona, USA, March 1999.

[12] Luotonen, A., *Web Proxy Servers*, Prentice Hall Computer Books, 1997.

[13] P. Maes, "Computational Reflection", Ph.D. Thesis, Technical Report 87-2, Artificial Intelligence Laboratory, Vrije Universiteit, Brussel, 1987.

[14] National Bureau of Standards, "Federal Information Processing Standard (FIPS) Publication 46: The Data Encryption Standard", 1977.

[15] National Institute of Standards and Technology, "Federal Information Processing Standard (FIPS) Publication 186: Digital Signature Standard", 1994.

[16] *OrbixWeb Reference Guide*, IONA Technologies PLC, 1998.

[17] Orfali, R. and D. Harkey, *The Essential Distributed Objects Survival Guide*, John Wiley, 1995.

[18] Pope, A., *The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture*, Addison-Wesley, 1998.

[19] R.L. Rivest "The MD5 Message Digest Algorithm", *RFC 1321*, MIT and RSA Data Security, Inc., April 1992.

[20] R.L. Rivest, A. Shamir, and L.M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, 1978, pp. 120-26.

[21] Sayood, K., *Introduction to Data Compression*, Morgan Kaufmann Publishers, 2000.

[22] J. Seitz, N. Davies, M. Ebner, A. Friday, "A CORBA-based proxy architecture for mobile multimedia applications", in *Proceedings of the 2nd IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS'98)*, Versailles, France, November 1998.

[23] M. Shapiro, "Structure and Encapsulation in Distributed Systems: the Proxy Principle", in *Proceedings of the 6th International Conference on Distributed Computing Systems*, Cambridge, Massachusetts, USA, May 1986, pp. 198-204.